Luis Garcia Remes

Game AI - Assignment 1

Kinematic Motion

For the basic motion part of the assignment I implemented a simple snap to target direction method and provided a sequence of velocities that the boid should move in to get the desired effect. Super basic and didn't really require a lot of work. Image A in the appendix shows the end position of the boid after it traverses the edge of the window.

Seek Steering Behaviors

For the "seek" steering part of the assignment I first implemented the seek behavior. In the book the behaviour is discussed as displaying an orbiting effect as it shoots past its intended target. However when I implemented it and selected a target on the screen the boid would shoot past some distance and then turn around and retrace its step but overshoot by a smaller direction. I never experienced creating a spiral pattern unless I clicked on the screen and waited for it to pick up speed and then click somewhere else that wasn't along its like of traversal. It would then just continue to go back and forth along the same traced path.

For the "arrive" behavior I followed the book originally but found I was getting the same behaviour as seek when finally reaching the target radius. The boid would periodically change its orientation and move a tiny bit outside the bounds of the target radius only to then move back. I found that the issue was that I wasn't setting the target velocity to 0 once inside the target radius instead the book pseudocode mentioned returning a null for steering. Which was causing an issue because it wasn't taking into account the need for a rectifying acceleration that would essentially compensate for our lack of a drag force.

I implemented 2 versions of this behaviour which can be seen by running my solution. In the first implementation I used a large slow radius and found that although it takes longer to get to the designated target it solved overshooting issues that were only apparent when going from one corner of the screen to the other. Basically the farther the starting position was to the target the more time the boid had to pick up speed which caused overshooting if the slow radius was too small. I also noticed some overshooting

issues when using the “look at where you are going” behaviour when selecting a target that was behind the boid (the opposite orientation of where it was facing). I think this was caused more or less because of the same basic idea as the linear overshooting problem. If the longer the more time the boid had to rotate before getting to the designated orientation allowed it to pick up speed and if the slow radius or its maxAngular was too large it would overshoot and then had to rectify itself. For the second version I use a smaller slow radius, larger maximum linear accel. I also use a larger max rotation and a smaller slow radius for “look where you are going”. Which is how I was able to notice the overshooting behaviour. Image B shows the first implementation and Image C shows the second. In image c you can see that the boid has overshot its orientation by a little bit. Because of this I would say my first implementation looks better but waiting for it to arrive is a little exhausting since it takes so long. So I would say my simulation could use a bit more tweaking.

Wander Steering Behaviors

For my “wander” behavior I implemented 2 versions. The first version has a small max angular acceleration and a “small” (at least compared to the second version) wander rate. In the second version I use a much larger max angular acceleration and a larger wander rate. The second version is very twitchy so I prefer the first version because the boid doesn’t look like a spaz. In my mind I wouldn’t really call the behaviour I am seeing as “wandering”. I guess when I imagine an enemy wandering in a game I imagine them walking in a direction stopping and then randomly deciding to move to another location inside some radius (the enemy would never walk out). Where as here it seems like the boid just keeps moving forward and then randomly decides to move a little bit from side to side. I imagine that I could get the behaviour I wanted by adding a timer so that the boid isn’t always trying to change its direction and adding a condition for the boid to stop moving if it wanted to and limiting its direction choice to always be inside a target radius.

I handled the boid going outside the default area of the window by teleporting it to the otherside when it crossed the boundaries that I set up. It is nothing fancy. I also use this method in the blocking simulation except that when the king boid crosses the boundary its teleported to the other side and its subjects(flock) are teleported to the other side while keeping their individual distance from the king the same.

Flocking Behavior and Blending/Arbitration

For the “flocking” behaviour I blended together “seek”, “velocity match”, and “separation” while my king boid wandered around. I ended up using a mix that relied primarily on “separation” then “seek” and got the least from “velocity match”. I also used “look where you are going” inorder to change their orientation. To get the displayed effect I had to use a decay coefficient of 500000.0f which seems a little weird to me but from discussing with other classmates they too had to use some rather large numbers.

I ran into one issue where my flock seemed to want to do nothing more than pile on top of each other and it didn’t matter how strong of a decay coefficient I chose. After a few hours and double checking the books pseudocode for separation I realized that the book had used the wrong math to calculate the direction of travel. Instead of using char.position - target.position to get a direction away from the target it was using target.position - char.postion so my boids where in fact being told to make a dog pile on the center of mass. So now that I have had multiple issues caused by incorrect pseudocodeI know to make sure the pseudocode makes sense instead of just accepting it as correct. I must have wasted at least an hour trying to play with the numbers to get the boids to separate using code that was telling them to congregate instead.

I initially simulated this using a flock of only 9 boids and it seemed to work just as well but it didn’t feel like a proper flock so I tripled the amount to what you can see simulated in the code now. There are still a few issues where it seems like everyone in a while two birds decide to chill next to each other and decided to ignore the separation behaviour and decide to only velocity match. I could not find weight values in the alloted time that got rid of that issue 100 percent of the time. You can see in images F and G that there are 2 boids that are almost on top of each other. I did not get a chance to simulate having two wanderers although I might try that now that the assignment is finished. I can imagine implementing a flock behavior in my thesis AI project where a group of lab rats follow the player in a flocking motion.

Additional Notes

I had never taken an AI class before and I didn’t realise that playing around with “magic numbers” and trial and error would be such a big part of implementing and creating the desired behavior for use in games. I spent about 2 hours playing with numbers for flocking alone. This is also the first time where I actually made use of pseudocode presented in a text book. Most of the programming classes I have taken (here at the U) have downplayed the use of pseudocode which seems weird to me since it sounds like most of the time when taking engineering tests for job applications specifically during interviews require clean and easy to follow pseudocode “skills”.

Appendix

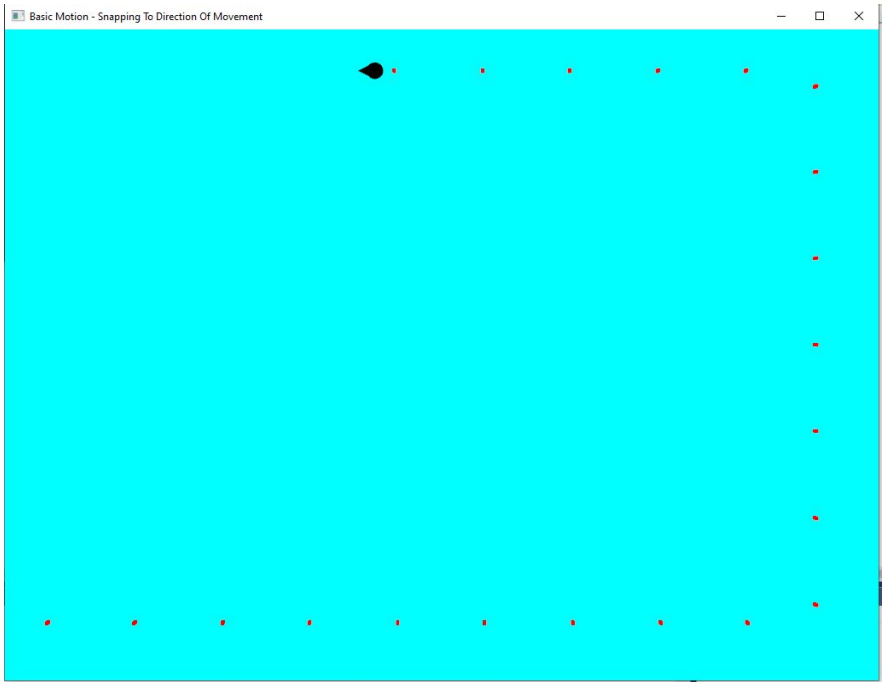

Image A: Basic Movement

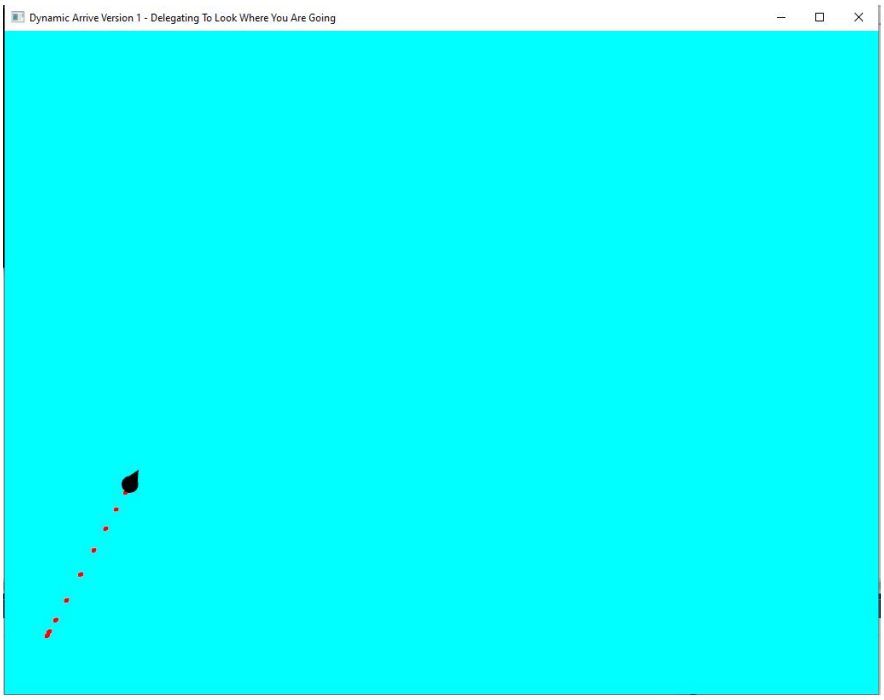

Image B: Dynamic Arrive Version 1

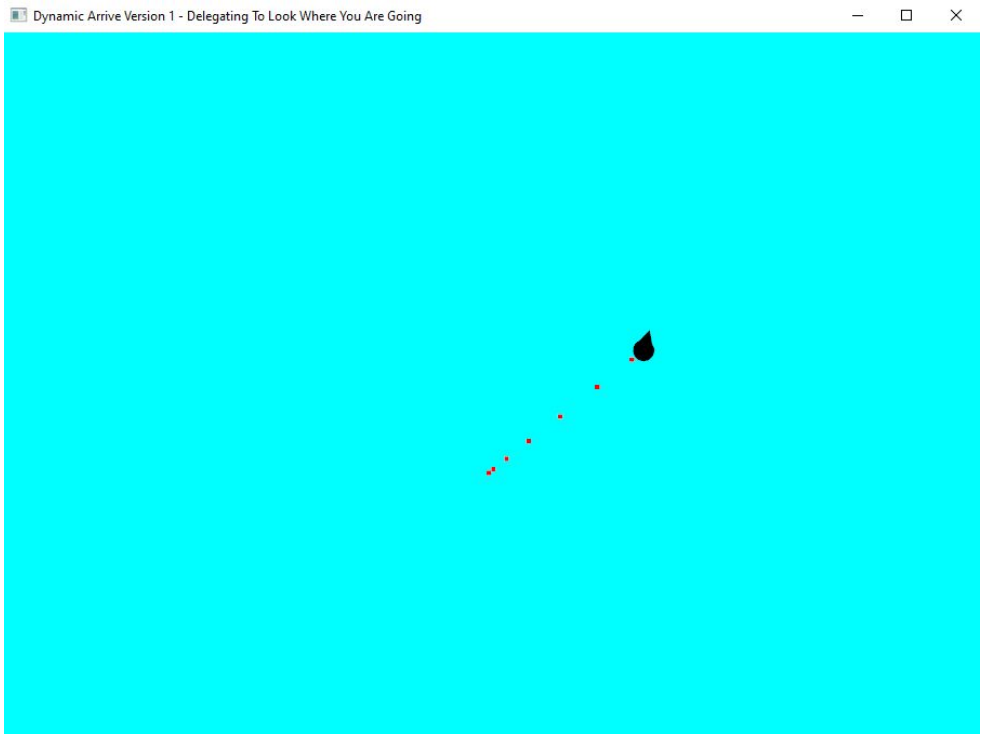

Image C: Dynamic Arrive Version 2

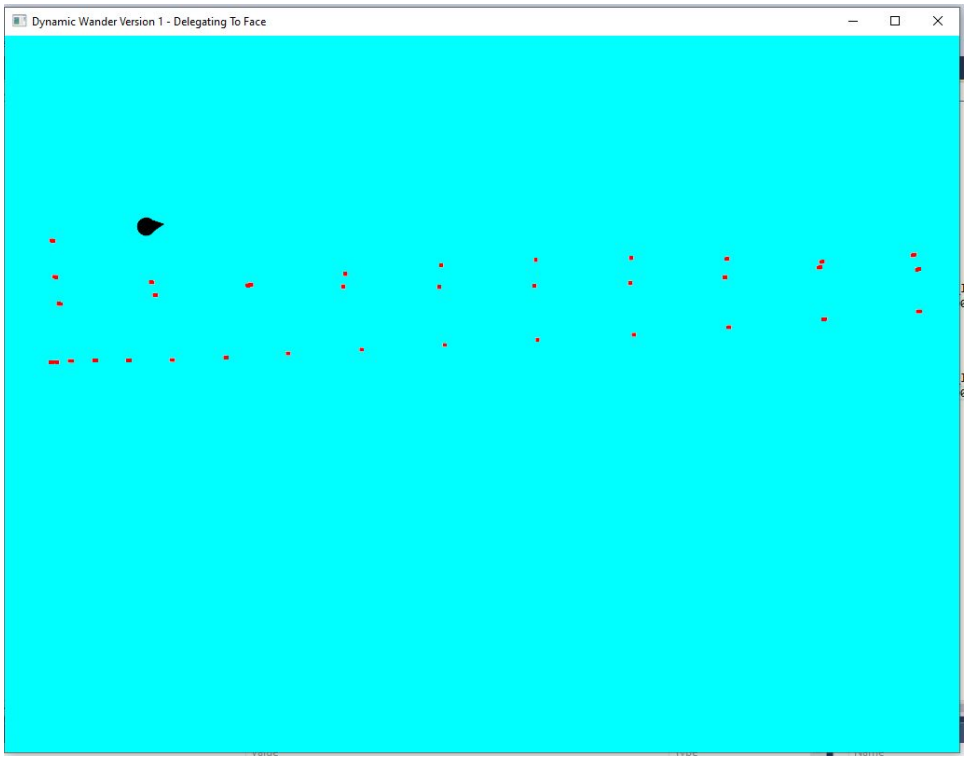

Image D: Dynamic Wander Version 1

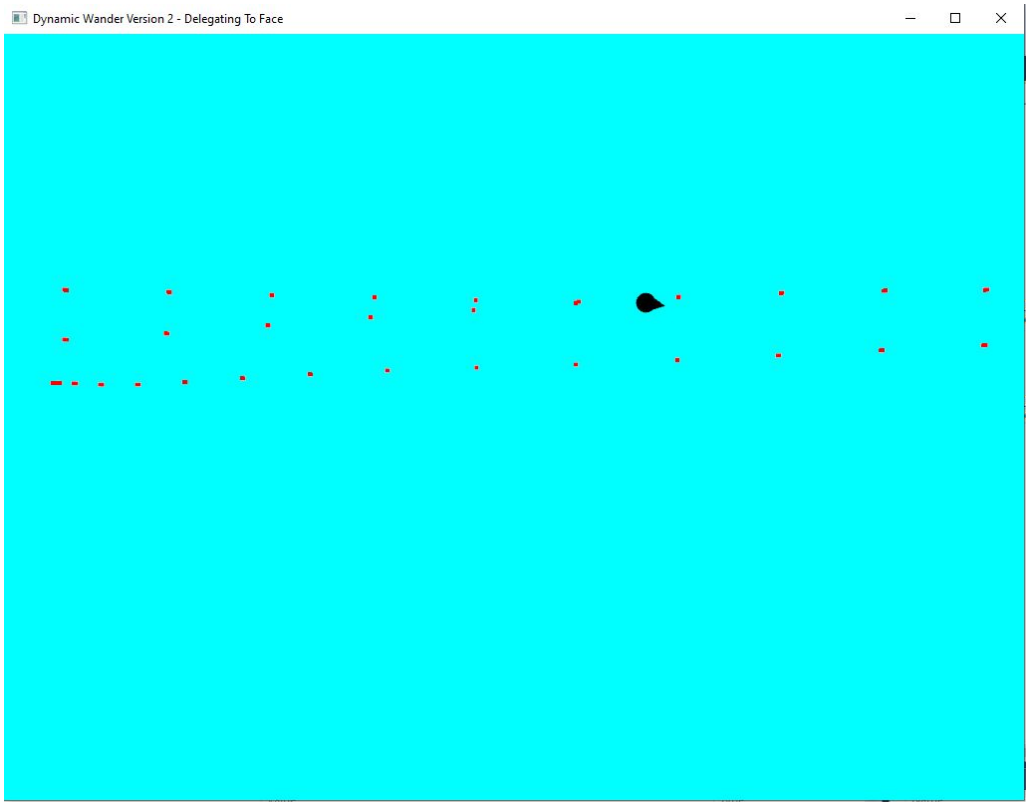

Image E: Dynamic Wander Version 2

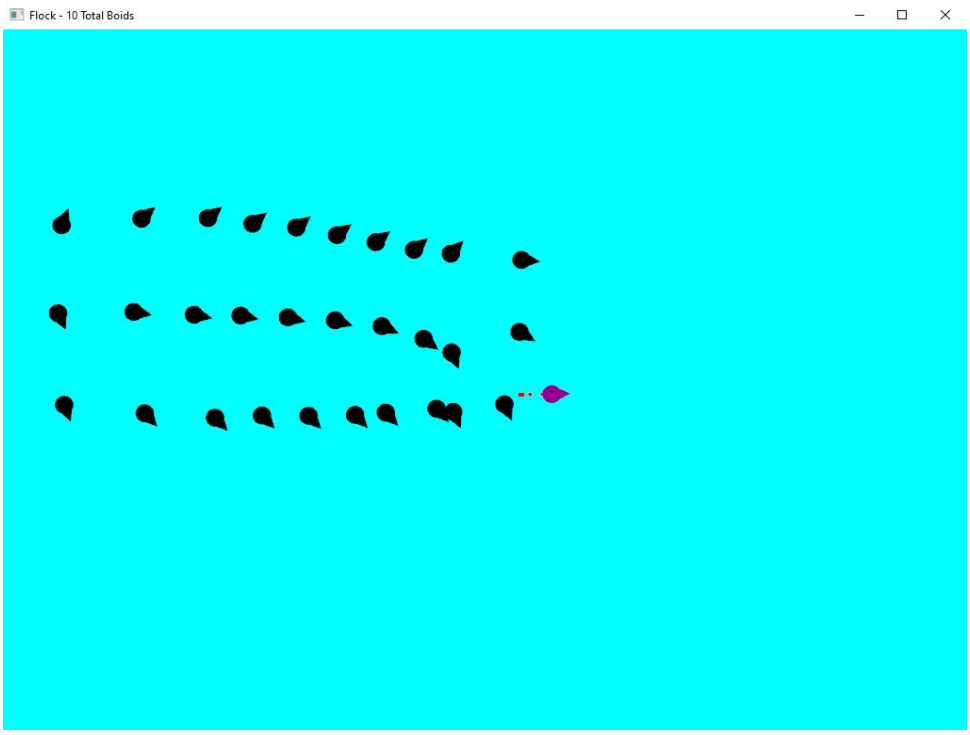

Image F: Flock Initial

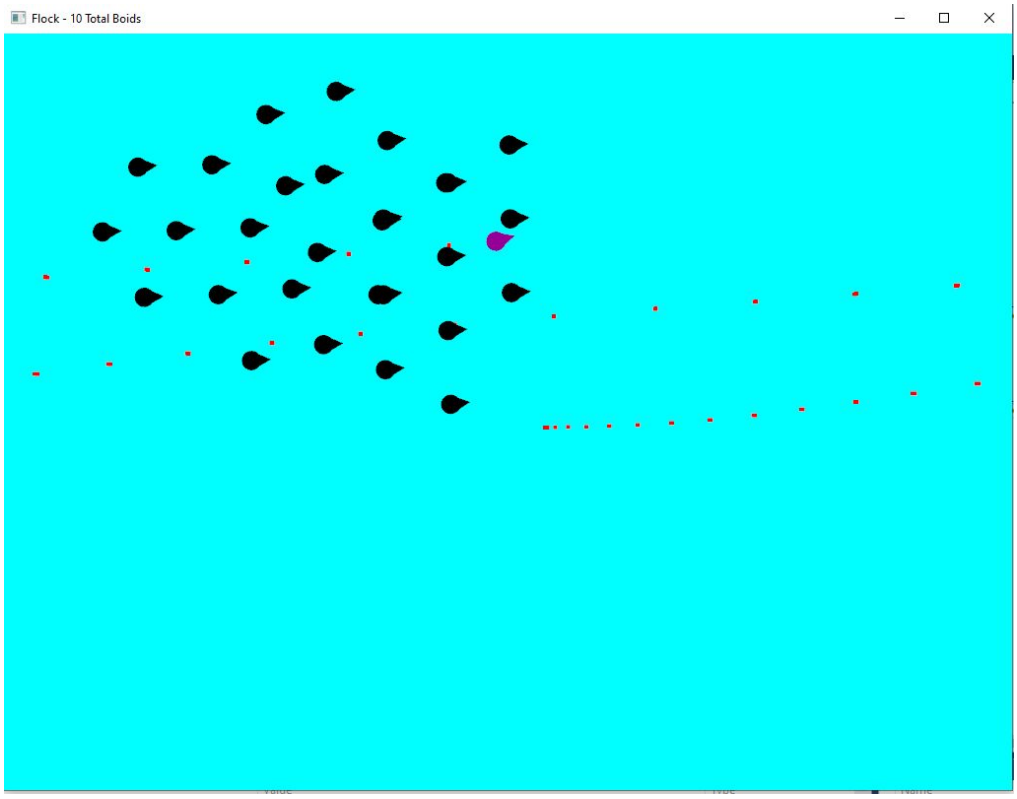

Image G: Flock After Some Time